Description

Method for specifying, executing and analyzing method sequences for recognition

The invention relates to a method for specifying, executing and analyzing method sequences for the recognition of dispatch labels and form entries.

The recognition is performed using OCR reading systems and video coding systems, where, if the reading results of the OCR reading system are not clear, the corresponding images of the dispatch labels or forms are sent to video coding stations where manual or partly manual coding with different coding steps is effected using databases for the labels. The labels to be read on the dispatches are largely addresses, and the databases used are address or forwarding dictionaries. The stipulated method sequence as the fundamental part of flow control is used to monitor and control the flow of processing for each individual dispatch or each form.

The method sequence defines for each dispatch or each form the sequence of the individual processing steps and the final decision regarding what needs to be done with the dispatch or with the form in the overall system. For all decisions, all information which is relevant to the decision and is available at this time is evaluated. Information relevant to the decisions is, by way of example, details about the type of sorting machine, the type of the individual coders and recognition results obtained to date. When all the necessary steps have been carried out, the results are sent to the sorting machine so that the dispatch or form can be distributed on the basis of this result.

To date, generic method sequences have been described

and implemented using a rule-based approach. Depending on the instance of application, hundreds of rules may therefore be needed in order to describe all the possible sequences. It has been possible to maintain these extensive rule mechanisms only with very great difficulty, however, and even small changes have had unforeseeable consequences or have resulted in a significant increase in the number of rules. Specifying and analyzing the rules for consistency and completeness has required a very high level of effort.

The invention is therefore based on the object of providing a method which greatly reduces the effort for specifying and analyzing generic method sequences. The invention achieves the object by means of the features of claim 1.

The invention's graphical representation in the form of a flowchart for the method sequences with attributes and function details, the automatic conversion into a loadable module which is called on the basis of the respective processing step, and the analysis using the clear representation significantly reduce the effort required for specifying and analyzing the method sequences, in particular in test phases.

Advantageous refinements of the invention are illustrated in the subclaims.

It is thus advantageous for value ranges and comments also to be entered in addition to the attributes.

It is also advantageous for not only the current values but also associated references relating to the names, comments and/or to the relevant elements/processing steps in the flowchart to be entered into the attribute file.

In another advantageous development, during analysis, the current method step is marked in the displayed flowchart.

For the sake of improved clarity, it is also advantageous for the attributes and comments of the current marked method step to be displayed.

The advantageous refinements above facilitate analysis of the method sequences.

It is also advantageous if, during the analysis, the parameters of the variables and/or functions are changed online and are entered into the flowchart.

The invention is explained in more detail below in an exemplary embodiment with reference to the drawings, in which

FIGURES 1a and b show a flowchart (part) as a graphical description of a method sequence for video coding a dispatch address, and

FIGURE 2 shows a screen display for offline analysis relating to the above flowchart.

The system described in the present case uses depicted flowcharts as a basis for providing an integrated graphical development environment for specifying, executing and analyzing method sequences for the recognition (automatic reading and/or video coding) of dispatch labels and form entries. This integrated graphical development environment comprises the graphical description of the sequences, production of the runtime environment and suitable tools for testing and diagnosing the sequences.

Specifically, the following functions are supported:

1. Recording and changing of the graphical description of the method sequence (in this case using Microsoft's Office drawing program VISIO),

2. Computer-aided automatic generation of program code (in this case C++ code) from the graphical description,

3. Tools for analysis (diagnosis and debugging) on the level of the original drawing (debugging and trace tools).

During overall processing, the flowcharts drawn remain the only program source, i.e. all the parts generated represent only a means to the end and do not need to be maintained by the user. The tools for analysis (diagnosis and debugging) likewise work with direct access to the graphical description.

The text below illustrates the method sequence for video coding dispatch addresses (coding strategy) as an example of application.

The coding strategy is described by one or more drawings on the basis of standardized drawing elements imitating ISO flowcharts. In addition, attributes and functions are defined in the drawings (e.g. in the form of special tables). Attributes and functions define the details of the influence of individual variables and their current values on the sequence of the coding strategy.

Attributes are defined by a name and a type. The types supported are "text" and "number". In addition, a value range and a comment can also be indicated for each attribute. The value range describes the set of

possible values for an attribute, and the comments should explain the significances of the attributes. This means that later it is possible to associate the comments with the current values automatically during analysis, which increases clarity enormously during analysis.

Attributes are either supplied to the coding strategy via the interface as input from the shell program or have a nature which is local to the coding strategy (e.g. for storing intermediate results).

In the case of functions, a distinction is likewise drawn between two types. Either functions are defined within the coding strategy (local functions) or they are part of the shell program and are called by the coding strategy with the current parameters.

FIGURE 1 shows, as an example, part of the method sequence (coding strategy) for video coding dispatch addresses.

An explanation of this is given in the table below.

Reference No	Description
1	Start the DECISION function; this is the main function of this coding strategy
2	Call the PROC EVENT function; this function is defined in the shell program (see comment 12)
3	Check whether RESULT STATUS attribute has been set
4	Jump to RETURN TO RIC continuation marker

5	Call the CHECK CODING function; ascertains what further coding steps need to be executed
6	Check whether RESULT STATUS attribute has been set
7	Jump to RETURN TO RIC continuation marker
8	Call the PP DECISION function; ascertains all the features of the sorting result
9	Check whether RESULT STATUS attribute equals RS_MORE CODING, i.e. checks whether further coding steps are necessary
10	Resets RESULT STATUS attribute
11	Jump to RETURN TO RIC continuation marker
12	List the functions of the shell program which are called from the coding strategy
13	RETURN TO RIC continuation marker
14	Call the TRACE VAR functions for the RESULT STATUS and UPDATE DSU attributes
15	Call the POST PROC function; processes statistics
16	Return to shell program; the next decision in the coding strategy can then be called

Table 1

The coding strategy fully described by the flowchart is read in and is converted into an internal representation (by VISIO using automation). From this internal representation, program code (C++ source code) is generated for the coding strategy. The C-compiler is then used to produce a loadable module which is called by the shell of the coding strategy. The coding

strategy is called for any changes to the attributes of a dispatch, and the rest of the sequence is redefined.

Besides the generated code for the actual method sequence, code for producing trace objects is additionally generated which controls the recording of diagnostic information during the time the coding strategy is running. The trace objects contain all the information relating to the attributes (current values, reference to names and comments) and a unique reference to the original elements of the flowchart.

While the coding strategy is being executed, for each call, these trace objects are stored together with the values of the attributes in a trace buffer per dispatch. If the trace function has been activated for the coding strategy, the trace buffers are stored in an attribute file for later analysis after a dispatch has been processed in full by the coding system. If the trace function is not active, the trace buffers are not stored.

In relation to the example considered in the present case (FIGURE 1), the text below shows parts of the generated C++ code.

Reference No

```
/*
 * generated code using template,
* template.cpp, do not edit
*/
#ifndef CS TEMPLATE_INCLUDED
#define CS TEMPLATE_INCLUDED
…
#include "tracebuf.h"
```

```
1                 void Decision (void)
                  {
                     TraceEntry (37,7,"Decision");
                     /*  The  main  entry  in  this
                     CodingStrategy */
                     TracePt (37,29);
2                    ProcEvent();
3                    if (ResultStatus. is_set()) {
4                    TracePt (37,36);
                     /*  Let  RIC  perform  next  coding
                     step/early decision */
                     /* Return to RIC */
                     TracePt (37,46);
                  L1:;
13                   TracePt (37,70);
14                   TraceVar (ResultStatus);
                  TraceVar (UpdateDSU);
                     TracePt (37,47);
15                   PostProc ();
                     TracePt (37,49);
16                   /*  Let  RIC  perform  next  coding
                     step/decision */
                  }  else {
                     TracePt (37,35);
                  L2:;
                     TracePt (37,9);
                     /* Determine, if additional coding
                     steps  are  necessary  before  final
                     decision can be made */
5                    CheckCoding ();
                     TracePt (37,16);
6                    if (ResultStatus.is_set()) {
                     TracePt (37,26);
                     …
7                    goto L1; /* 37,46 */
                     } else {
                     TracePt (37,18);
                     …
```

```
8                   PPDecision ();
                    TracePt (37,25);
9                   if (ResultStatus == RS_MoreCoding){
                    TracePt (37,27);
10                  ResultStatus.clear ();
                    …
                    goto L2;  /* 37,35 */
                    } else {
11                  TracePt (37,24);
                    …
                    goto L1; /* 37,46 */
                        }
                    }
                  }
                TraceExit ("Decision"); /* 37,7 */

            }
```

The trace buffer data recorded in the attribute file can be used for detailed analysis of the flow, controlled by the coding strategy, of each dispatch's processing.

A special user interface is used to show the user the information from the recorded trace buffer together with the associated original flowchart for the coding strategy.

During the offline analysis, it is possible to reconstruct the flow of all the processing steps for a dispatch. In this context, the steps executed are displayed to the user in the original flowchart for the coding strategy with all the information relevant to analysis. To analyze the sequences of a coding strategy, the following information is displayed to the user in individual windows, as shown in figure 2:

- drawing of the flowchart, the current position within

the original drawing of the coding strategy is marked, 20,

- the image of the dispatch, 21,
- information relating to the attributes (based on the processing step, the particular current values of the attributes are displayed), 22,
- call list for function calls, 23,
- section index, number of the current section in the drawing (section and subsection for the current position) and list of all sections, 25,
- optionally additional Logfile outputs, 24.

The user now has the opportunity to reconstruct the flow, stored in the trace buffer, of the respective dispatch's processing. In this case, as with a source level debugger, he has the opportunity to move through the trace buffer.

The normal commands of a source level debugger are supported:

- use of break points,
- step in,
- step over,
- step out, etc.

In the graphical representation of the coding strategy (VISIO flowchart), the particular current step is marked. The user sees the marker being guided through the drawing as the individual steps are executed. With each step, values and comments relating to the current values are displayed for all attributes. This means that the user has available, for each recorded dispatch, all the information necessary for analyzing the sequences of the coding strategy, i.e. all sequences can subsequently be analyzed in detail.

This method affords the opportunity for detailed analysis in relation to each individual dispatch which

has been processed in the system. Even after a very large number of dispatches has been processed, analysis can still be carried out hours after the dispatches have actually been processed. This diagnostic tool allows even highly complex coding strategies to be analyzed, and in the case of incorrect decisions this method can be used to ascertain the specific causes of error.

The method described is also suitable for online tests. With online tests, a coding strategy under development can be executed and tested step by step. In this context, the user can use the development environment's user interface to define a break point in the depicted coding strategy with a special command. When the break point is reached, the image of the dispatch currently being processed is turned on in the user interface and the current trace buffer is used as described above to display and provide a detailed analysis of the flow up to the current situation. Unlike in the case of offline analysis, the attributes can also be changed in addition in this case, and hence the rest of the flow can be influenced. After analysis, processing can be continued (possibly up until the next break point).